Trie字典树

2019年5月14日 19:36

1. 前缀字典树：快速检索前缀的多叉树存储结构，每个结点又有指向下一字符的指针
   1) 初始化：仅含根结点，各字符指针都指向空
   2) 插入字符串：先让指针P=根结点，然后遍历每个c
      i. 如果P已经指向过C，就让P=从P指向C的Q
      ii. 如果P没有指向Q就新建一个Q，再让P指向Q
      iii. 遍历到底后加一个末尾标记
   3) 检索：让P从根节点扫描每个c，并转向下一个Q，能找到末尾标记说明存在
   4) 设结点数为N，字符集大小为C则空间复杂度为O（NC）
2. 以下标作为“指针”，0为空，1为根，其他结点依次编号的trie
   1)
```
const int MN = 1000005;
int trie[MN][26];
int tot= 1;
bool ed[MN];
//string ts[MN];//调试用
```
   2)
```
void insert(char*s){
        int p= 1;
        int len= strlen(s);
        for_(i,0,len){
                int ch= s[i]-'a';
                if(!trie[p][ch])
                        trie[p][ch]= ++tot;
                //ts[trie[p][ch]] = ts[p] + s[i];
                p= trie[p][ch];}
        ed[p]= 1;}
//for__(i,1,tot)
//        cout<<ts[i]<<"是结尾的字符串数："<<ed[i]<<endl;
```
   3)
```
bool search(char*s){
        int p= 1;
        int len= strlen(s);
        for_(i,0,len)
                if(!(p= trie[p][s[k]-'a']))
                        return 0;
        return ed[p];    //查到尾也不一定return 1，因为可能查到的是子串
}
```
3. 任板的动态内存字典树
```
struct node
{
  node* nxt[2];
  int val,dep;
}head;

int ans,n;
char s[205];

void del(node* p)
{
  if(p->nxt[0]) del(p->nxt[0]);
```

```
    if(p->nxt[1]) del(p->nxt[1]);
    if(p!=&head) free(p);
}

node* newnode(int d)
{
    node* p=(node*)malloc(sizeof(node));
    p->dep=d+1;
    p->val=0;
    p->nxt[0]=p->nxt[1]=0;
    return p;
}

void trie_insert(char s[],int l)
{
    node* p=&head;
    for(int i=0;i<l;i++)
    {
        int c=s[i]-'0';
        if(p->nxt[c]==NULL) p->nxt[c]=newnode(p->dep);
        p=p->nxt[c];
        p->val++;
        ans=max(ans,p->dep*p->val);
    }
}

void solve()
{
    head.val=head.dep=0;
    head.nxt[0]=head.nxt[1]=0;
    ans=0;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%s",s);
        trie_insert(s,strlen(s));
    }
    del(&head);
    printf("%d\n",ans);
}
```

- ◆
- ◆ 例

4. 最大公共前缀长度在[l,r]的有序串对数
    1) 引：相同串的最大公共前缀长度即为字符串长度，即**都长为d**
    2) 引：最大公共前缀长为d但**不都长为d**的有序串对数量=Σ_{包含这个长为d的前缀的字符串S}不与S相等的字符串数量，这个不相等可以通过指向不同子树来解决
    3) 为了快速求区间和，事先算好前缀和。顺带一提居然不爆int
    4)
```
const int MN= 1000005;
int n,m,k,t;
int tot= 1;
int trie[MN][30];          //字典树
int pss[MN],ed[MN];    //经过该结点和在该结点结尾的串数量
int sum[105];   //f的前缀和
```

```
string ts[MN];
```

5)
```
cin>>n>>m;
string s;
for_(i,0,n){
    cin>>s;
    int p= 1;           //初始结点
    for(auto c : s){
        int nxt= c-'A';
        if(!trie[p][nxt])
            trie[p][nxt]= ++tot;
        //ts[trie[p][nxt]] = ts[p]+ c;
        ++pss[p];
        p= trie[p][nxt];}
    ++pss[p];       //这段不能删，会让比它长的字符串不好转移
    ++ed[p];}
//cout<<endl<<"trietree:\n";
//for__(i,1,tot)
//      cout<<ts[i]<<"\t经过串数："<<pss[i]<<"\t结尾串数："<<ed[i]<<endl;
queue<int>q;  //广搜用的层队列
q.push(1);
for__(dep,0,100){       //遍历每层
    int sz= q.size();
    for_(j,0,sz){
        int p= q.front();
        q.pop();
//          cout<<"dep"<<dep<<" :"<<ts[p]<<endl;
        sum[dep] += ed[p]* ed[p];       //(p结尾,p结尾)
        for_(nxt,0,26){
            int k= trie[p][nxt];
            if(k)
                sum[dep]+= pss[k]* (pss[p]- pss[k]),
                //(k子树,其他子树)
                q.push(k);}}
    sum[dep+1]= sum[dep];}
//for_(i,1,5)
//      cout<<"f("<<i<<")="<<sum[i]-sum[i-1]<<endl;
int l,r;
for_(i,0,m)
    cin>>l>>r,
    cout<<sum[r]-sum[l-1]<<'\n';
```

5. L语言：找字典上能理解的无空格语句最长前缀（P2292）
   1) 空开序号0，初值令序号0可以是结尾，当序号i可以是结尾时，在字典树上j循环找i+1开始的字符串，每次找到结尾时令j可以是结尾，字典中找不到j时重新回去找下一个可以是结尾的i
   2)
```
int p;  //临时结点指针
char s[MN];     //字符串，空出0
int trie[200][26];
bool ed[200*26];
bool b[MN];     //备选答案，序号i可作为结尾时，b[i]=1
```
   3)
```
for_(i,0,n){
```

```
        p= 1;
        cin>>s+1;
        len= strlen(s+1);
        for__(i,1,len){
            int c= s[i]- 'a';
            if(!trie[p][c])
                trie[p][c] = ++tot;
            p= trie[p][c];}
        ed[p] =1;}
    while(m--){
        cin>>s+1;
        len= strlen(s+1);
        ms(b,0);
        b[0] =1;            //0是答案的初值
        for_(i,0,len)
            if(b[i]){  //i可以作为结尾被找到，即i+1可以是下一个开头
                p =1;
                for__(j,i+1,len){
                    p= trie[p][s[j] -'a'];
                    if(!p)    //这个结点不在字典
                        break;
                    if(ed[p])
                        b[j] =1;}}
        rof__(i,len,0)
            if(b[i]){
                cout<<i<<'\n';
                break;}}
```

6. 第k小异或数
    1) 引：两操作数相同时异或结果为0，不相同时异或结果为1
    2) 引：求x的最小异或数：把01串从高位到低位存到字典树上，如果能在树上找到x对应的路径，说明能异或出0，输出0即可；如果不能，则只给找不到相同路径的层对应位赋1即可
    3) 转：求x的第k小异或数：将每个结点为根的子树大小记下来；如果当前层存在x对应位的值，且沿着当前结点有k种或以上个子结点，说明这一位为0时可以找到k种异或结果，继续往下找即可；对应位的子孙结点没有k个时，说明这一位的异或结果为0时找不到k个数，所以正确答案里，这位应为1，接下来应向相反位的方向，且顺着往下找第（k-对应位子结点数）小的数
    4) 注意：结点数量最坏情况大概是左操作数的数量*位数，字典树的第一维和子树大小记录都应往大了开
    5)
```
const int MN= 10005;
int trie[70*MN][2];       //字典树
int tot= 1;                                  //字典树结点量
int cnt[70*MN];           //各结点为根的子树大小

void insert(ll x){        //把x的低61位插入字典树
    int p= 1;                                //初始结点下标
    rof__(i,60,0){
        int s= x>>i & 1;  //下一结点对应的0或1
```

```
        if(!trie[p][s])                 //不存在该结点，新分配一个下标给它
            trie[p][s]= ++tot;
        p= trie[p][s];
        ++cnt[p];}}

ll search(ll x, int k){         //找x在字典树上的第k小异或值
    ll ret= 0;                          //待return值，初值赋0
    int p= 1;                           //初始结点下标
    rof__(i,60,0){
        int s= x>>i & 1;
        if(cnt[trie[p][s]] >= k)    //树上存在前i位相同的路径
            p= trie[p][s];                  //继续往下找即可
        else     //不存在，或存在但不够k个数，此时该位需赋1
            k= k- cnt[trie[p][s]],      //先用指针更改目标k
            p= trie[p][1-s],            //再给结点指针转向
            ret+= 1LL<<i;                   //再给返回值第i位赋1
    }
    return ret;}
```

6)
```
ll a,b[MN];
int c;
int n,m;
cin>>n>>m;
for_(i,0,n)
    cin>>a,
    insert(a);
for_(i,0,m)
    cin>>b[i];
for_(i,0,m)
    cin>>c,
    cout<<search(b[i],c);
```